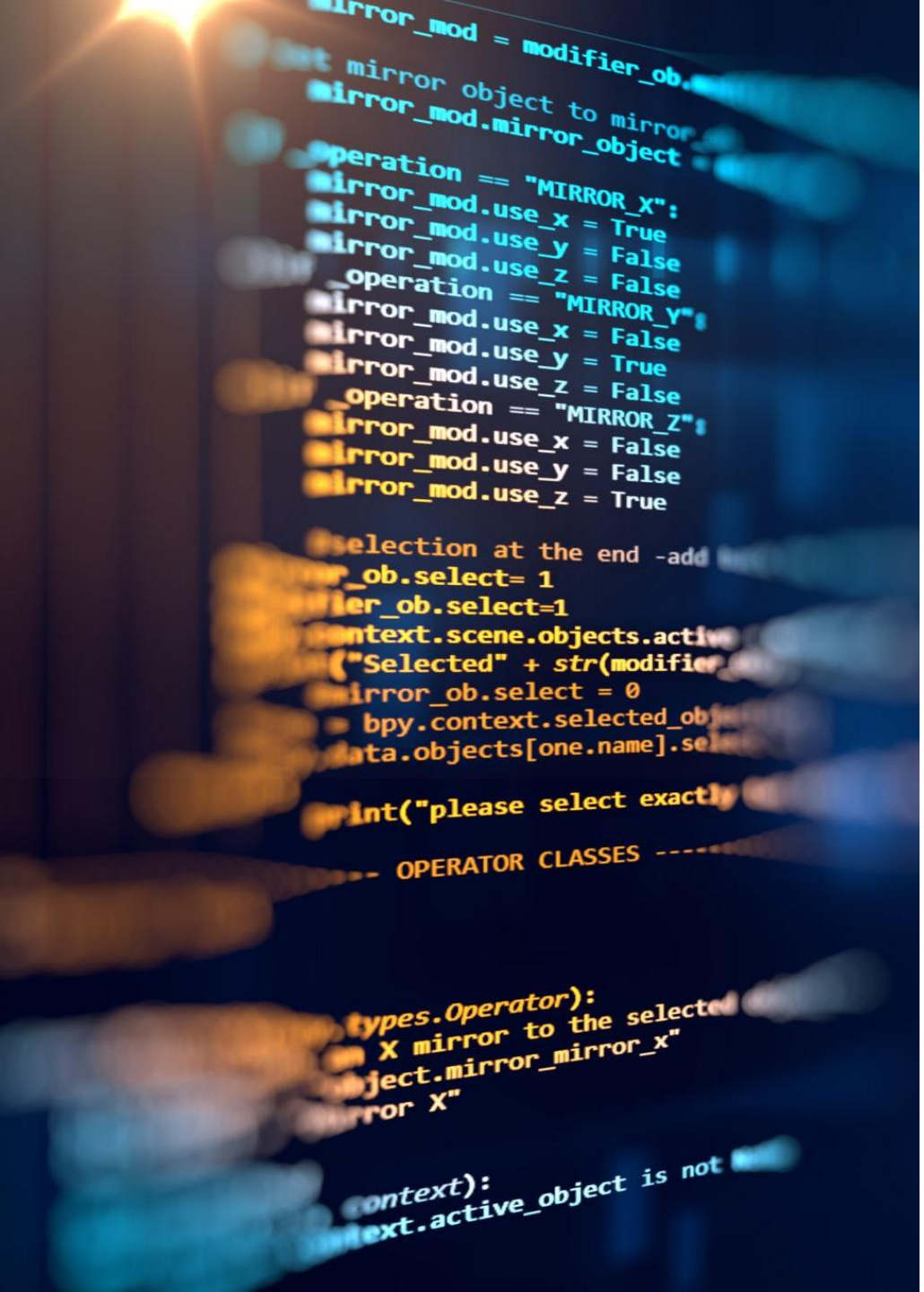

# Programmieren lernen mit Java

Said Cetin

# Kapitel 2 Agenda

1. Codeblöcke mit {}
2. Die if-Anweisung
3. Die if-else-Struktur
4. Mehrere Bedingungen mit else if
5. Das Dangling-else-Problem
6. Die switch-Anweisung
7. Default im switch
8. Break im switch

# Thema 1: Codeblöcke mit {} in Java

Agenda

- Was ist ein Block?
- Warum sind Blöcke wichtig?
- Beispiele für Codeblöcke
  - Beispiel 1
  - Beispiel 2
  - Beispiel 3
  - Beispiel 4
- Merksätze & Zusammenfassung
- Verständnisfragen (zur Abfrage)

# Was ist ein Block?

Ein Block ist eine Gruppe von Anweisungen, die in geschweiften Klammern {} eingeschlossen ist.

In Java sind Blöcke ein zentrales Element der Programmlogik

Sie kommen überall vor: bei if, Schleifen, Methoden, Klassen usw.

Syntax :

```
{
  // Anweisung
}
```

# Wofür braucht man Blöcke?

- Um mehrere Anweisungen logisch zusammenzufassen
- Um festzulegen, welche Anweisungen zu einer Bedingung oder Schleife gehören.
- Um den Sichtbarkeitsbereich ( Scope ) von Variablen zu steuern
- Um Code lesbar und strukturiert zu halten

# Beispiel 1: mit und ohne Block

```
If (x > 0){
        System.out.println(“x ist  positiv“);
        x++;
}

If(x > 0)
        System.out.println(“x ist  positiv“);
        System.out.println(“Diese Zeile gehört nicht zur Bedingung!“);
```

# Beispiel 2:  Fehler durch fehlende Klammern

- Einrückung spielt **keine Rolle** in Java, nur {} sind entscheidend!
- Dadurch entstehen oft **logische Fehler**, die schwer zu erkennen sind.
- Best Practise: Immer Klammern {} verwenden, auch bei nur einer Anweisung

# Beispiel 3: Sichtbarkeit von Variablen

```
{
   int a = 10;
   System.out.println(a);   // gültig
}
System.out.println(a);                    // Fehler! a existiert nur im Block
```

- Variablen existieren **nur im Block**, in dem sie deklariert wurden.
- **Außerhalb** sind sie nicht mehr sichtbar -> Kompilierfehler.

# Beispiel 4: Wo kommen Blöcke überall vor?

- **Methoden:**

```
public void hallo() {
   System.out.println("Hallo");
}
```

- **Schleifen:**

```
for (int i = 0; i < 5; i++) {
   System.out.println(i);
}
```

- **Bedingungen:**

```
if (x > 0) {
   ...
}
```

- **Klassen, statische Blöcke, Initialisierer etc.**

# Merksätze & Zusammenfassung

- Ein Block beginnt mit {  und endet mit  }
- Variablen gelten nur innerhalb des Blocks, in dem sie deklariert wurden
- Einrückung ist optisch hilfreich, aber für Java nicht relevant
- Auch bei einer Anweisung: immer {} setzen, um Fehler zu vermeiden.

# Verständnisfrage zur Abfrage

- Was ist ein Block in Java, und wie wird er geschrieben?
- Was passiert, wenn man Klammern {} weglässt?
- Warum führt Einrückung allein nicht zur Gruppierung?
- Kann man eine Variable außerhalb ihres Blocks verwenden?
- Nenne mindestens zwei Stellen im Code, an denen man Blöcke verwendet.

# Thema 2 : Die if – Anweidung in Java

Agenda

- Was ist eine if -Anweisung?
- Beispiel für eine einfache if-Anweisung?
- Bedingungen mit Vergleichsoperatoren
- If mit einer einzigen Anweisung
- Kombination mit logischen Operatoren
- Best Practices für if
- Verständnisfragen

# Was ist eine if- Anweisung?

- Mit einer if Anweisung kann ein Programm Entscheidungen treffen
- Sie prüft eine Bedingung, die ein true oder false ergibt
- Wenn die Bedingung wahr (true) ist, wird der folgende Codeblock ausgeführt

Syntax:

```
if (Bedingung) {
   // Anweisungen
}
```

# Beispiel für eine einfache if-Anweisung

```
int note = 1;

if (note == 1) {
   System.out.println("Sehr gut!");
}
```

- Ausgabe -> Sehr gut! , nur wenn note gleich 1 ist.

# Bedingungen mit Vergleichsoperatoren

Die Bedingung muss immer einen boolean-Wert ergeben

| Ausdruck | Bedeutung |
| --- | --- |
| x == y | Gleich |
| x != y | Ungleich |
| x > y | Größer als |
| x < y | Kleiner als |
| x >= y | Größer gleich |
| x <= y | Kleiner gleich |

# If mit einer einzigen Anweisung

```
if (x > 0)
   System.out.println("x ist positiv");
```

- Nur eine einzige Anweisung wird ausgeführt. Keine {} nötig.

Besser:

```
if (x > 0) {
   System.out.println("x ist positiv");
}
```

Sicherer, übersichtlicher, erweiterbar. ✅

# Kombination mit logischen Operatoren

```
int a = 5;
int b = 10;

if (a > 0 && b > 0) {
   System.out.println("Beide Zahlen sind positiv");
}
```

`&&` UND- Verknüpfung -> beide Bedingungen müssen wahr sein.

Andere Operatoren:

- `||` -> ODER
- `!` -> NICHT

# Best Practises für if

- Verwende immer {} , selbst bei einer Anweisung
- Halte Bedingungen klar und lesbar
- Nutze sprechende Variablennamen
- Nutze logische Operatoren bewusst (nicht zu verschachtelt!)

# Verständnisfragen

- Was  passiert, wenn eine if- Bedingung nicht erfüllt ist?
- Muss eine if- Anweisung immer einen Block enthalten?
- Welche Werte darf  eine Bedingung in Java zurückgeben?
- Was ist der Unterschied zwischen == und = in einer Bedingung?

# Thema 3 : Die if–else Struktur in Java

Agenda

- Was ist if -else?
- Einfaches Beispiel mit if-else?
- Wichtiges zur Ausführung
- Gültigkeit von Variablen
- If-else mit Methoden oder logischen Ausdrücken
- Best Practices zu if-else
- Verständnisfragen

# Was ist if-else?

- If-else wird verwendet, wenn zwei alternative Wege möglich sind
- Entweder wird der if-Block oder der else-Block ausgeführt. **niemals beide!**
- Es entscheidet die Bedingung im if

**Syntax :**

```
if (Bedingung) {
   // wenn true
} else {
   // wenn false
}
```

# Einfaches Beispiel mit if-else

```
int zahl = 5;


if (zahl % 2 == 0) {
   System.out.println("Gerade Zahl");
} else {
   System.out.println("Ungerade Zahl");
}
```

- Wenn zahl durch 2 teilbar ist -> „Gerade Zahl“ ,
  sonst -> „Ungerade Zahl“

# Wichtiges zur Ausführung

- Nur eine der beiden Blöcke wird ausgeführt
- Wenn if true -> nur der obere Block
- Wenn if false -> nur der else-Block

- ❗ ❗ **Es kann nie beides gleichzeitig laufen** ❗ ❗

# Gültigkeit von Variablen

```
if (x > 10) {
   int a = 5;
   System.out.println(a);
} else {
   // System.out.println(a);          // Fehler a existiert hier nicht
}
```

Variablen, die **im Block** deklariert werden, gelten **nur dort**.

# If-else mit Methoden  oder logischen Ausdrücken

```
int alter = 20;

if (istVolljährig(alter)) {
   System.out.println("Zutritt erlaubt");
} else {
   System.out.println("Zutritt verweigert");
}

public static boolean istVolljährig(int a) {
   return a >= 18;
}
```

Auch **Methodenaufrufe** und **logische Ausdrücke** können als Bedingung verwendet werden

# Best Practices zu if-else

- Halte Bedingungen einfach und lesbar.
- Verwende Klammern `{}` immer, auch bei einer Anweisung
- Nutze else nur wenn nötig. Manchmal reicht ein if

# Verständnisfragen

- Wird bei if-else immer beides ausgeführt?
- Was pasiert, wenn die if-Bedingung false ist?
- Warum sollte man auch bei einer Zeile {} setzen ?
- Was passiert mit Variablen, die nur im if- Block deklariert wurden ?
- Kann man in der Bedingung Methoden aufrufen?

# Thema 4: else if – Mehrere bedingungen prüfen

Agenda

- Was ist else if?
- Beispiel – Altersabfrage
- Reihenfolge ist entscheidend
- else if mit mehreren Bedingungen
- Alternativen zu else if
- Best Practices zu else if
- Verständnisfragen

# Was ist else if

- else if erlaubt es, **mehrere Bedingungen nacheinander** zu prüfen
- Wird die erste if-**Bedingung nicht erfüllt**, wird die nächste geprüft usw.
- **Sobald eine Bedingung** true ist, wird ihr Block ausgeführt.
  Danach wird **alles Weitere übersprungen.**

# Beispiel- Altersabfrage

```
int alter = 70;

if (alter < 18) {
   System.out.println("zu jung");
} else if (alter <= 65) {
   System.out.println("erwachsen");
} else {
   System.out.println("Rentner");
}
```

Nur **eine einzige Ausgabe** wird erzeugt, je nachdem, welche Bedingung zutrifft

# Reihenfolge ist entscheidend

- Beispiel mit falscher Reihenfolge:

```
int alter = 16;


if (alter <= 65) {
   System.out.println("erwachsen");
} else if (alter < 18) {
   System.out.println("zu jung");
}
```

Ausgabe: ????

Wann wird die zweite Bedingung erreicht?

# else if mit mehreren Bedingungen

- Kombination mit logischen Ausdrücken

```
if (punktzahl >= 90) {
   System.out.println("Sehr gut");
} else if (punktzahl >= 75) {
   System.out.println("Gut");
} else if (punktzahl >= 60) {
   System.out.println("Ausreichend");
} else {
   System.out.println("Nicht bestanden");
}
```

Bewertung erfolgt stufenweise nach Punkten

# Best Practices für else if

- Prüfe vom spezifischsten zum allgemeinsten
- Achte auf überschneidende Bedingungen
- Setze Klammern `{}` immer
- Nutze else am Ende für **Standardfall**

# Verständnisfragen

- Was passiert, wenn mehrere Bedingungen zutreffen?
- Warum ist die Reihenfolge bei else if wichtig?
- Wird bei else if mehr als ein Block ausgeführt?
- Wozu dient das letzte else?
- Wann sollte man lieber switch statt else if verwenden?

# Thema 5: Das Dangling-else-Problem

Agenda

- Was ist das Dangling-else-Problem?
- Beispiel für das Problem
- Korrekte Lösung mit Klammern
- Alternative Interpretation
- Best Practices zur Vermeidung
- Verständnisfragen

# Was ist das Dangling-else-Problem

- In verschachtelten **if**-Anweisungen kann nicht eindeutig zugeordnet werden, zu welchem **if** ein **else** gehört.
- Java ordnet ein **else** immer dem nächsten offenen **if** zu
- Das kann zu unerwartetem Verhalten führen.

# Beispiel für das Problem

```
int a = 5, b = -2;


if (a > 0)
   if (b > 0)
      System.out.println("Fall A");
   else
      System.out.println("Fall B");
```

Welche Bedingung gehört zum else?

- zum Inneren if
- zum äußeren if

# Korrekte Lösung mit Klammern

```
if (a > 0) {
   if (b > 0) {
      System.out.println("Fall A");
   } else {
      System.out.println("Fall B");
   }
}
```

Jetzt ist klar: **else** gehört zum **inneren if**.

Durch {} wird die **Logik eindeutig**

# Alternative Interpretation

```
if (a > 0) {
   if (b > 0)
      System.out.println("Fall A");
}
else {
   System.out.println("Fall B");
}
```

Jetzt gehört das **else** zum äußeren **if** – dank der Klammerstruktur

# Best Practices zur Vermeidung

- **Immer Klammern setzen**, selbst bei nur einer Anweisung
- Nutze **Einrückung konsequent**, aber verlasse dich nicht darauf!
- Halte verschachtelte Bedingungen **so flach wie möglich**

# Verständnisfragen

- Warum ist das Dangling-else-Problem problematisch?
- Was macht Java, wenn es mehrere **if** und ein **else** gibt?
- Wie kannst du eindeutig zeigen, zu welchem **if** ein **else** gehört?
- Was unterscheidet **Einrückung** von **Blockstruktur** in Java?
- Kann das Problem auch bei anderen Sprachen auftreten

# Thema 6: Die switch-Anweisung in Java

Agenda

- Was ist die switch-Anweisung?
- Beispiel: Wochentag ausgeben
- Typen für switch in Java
- Was passiert ohne break?
- Best Practices für switch
- Verständnisfragen zu switch

# Was ist die switch-Anweisung?

- Switch ist eine Alternative zu vielen if-else-Anweisungen
- Sie wird verwendet, um **eine Variable gegen mehrere festen Werte** zu vergleichen.
- Besonders nützlich bei klar abgegrenzten Fällen (z.B. Menüs, Tage, Note).

**Syntax:**

```
switch (variable) {
  case wert1:
    // Anweisungen
    break;
  case wert2:
    // Anweisungen
    break;
  default:
    // Standardfall
}
```

# Beispiel: Wochentag ausgeben

```
int tag = 2;


switch (tag) {
  case 1:
    System.out.println("Montag");
    break;
  case 2:
    System.out.println("Dienstag");
    break;
  case 3:
    System.out.println("Mittwoch");
    break;
  default:
    System.out.println("Ungültiger Tag");
}
```

# Typen für switch in java

- Java erlaubt **switch** für:
  - **Int**, **byte**, **short**, **char**
  - **String** (seit Java 7)
  - **Enum** (Aufzählungstypen)
  - **ab Java 14**: **switch** als Ausdruck mit Rückgabewert (optional)

- Nicht erlaubt: **boolean**, **double**, **float**, Objekte (außer **String**)

# Was passiert ohne break?

```
int x = 2;


switch (x) {
   case 1:
      System.out.println("Eins");
   case 2:
      System.out.println("Zwei");
   case 3:
      System.out.println("Drei");
}
```

Ausgabe:

Zwei

Drei

Ohne break wird einfach weiter ausgeführt. Das nennt man Fall Through

# Best Practices für switch

- Verwende immer **break**, um Fall-Through zu vermeiden
- Nutze **default**, um **alle anderen Fälle abzufangen**.
- Verwende **switch**, wenn du viele **gleichartige Vergleiche** hast.
- Nutze **if bei Bereichen**, logischen Ausdrücken oder Methoden.

# Verständnisfragen zu switch

- Wann ist switch sinnvoller als if-else?
- Was passiert, wenn man break vergisst?
- Wozu dient der default – Fall?
- Welche Datentypen dürfen in switch verwendet werden?
- Wie viele case-Zweige kann ein switch haben?

# Thema 7 & 8 : default und break in switch-Anweisungen

Agenda

- Was ist default im switch?
- Was ist break im switch
- Sonderfall – bewusstes Durchlaufen
- Best Practices für switch
- Verständnisfragen zu default und break

# Was ist default im switch?

- Default wird ausgeführt, wenn kein case zutrifft
- Er ist optional, aber sehr empfohlen
- Default kann überall im switch stehen, meist aber am ende.

```
int zahl = 5;


switch (zahl) {
   case 1:
      System.out.println("Eins");
      break;
   case 2:
      System.out.println("Zwei");
      break;
   default:
      System.out.println("Ungültige Eingabe");
}
```

# Was ist break im switch?

- Break beendet die Ausführung innerhalb des switch.
- Ohne break wird weiter zum nächsten Fall gesprungen (Fall-Through)
- Das kann absichtlich oder versehentlich passieren

```
int x = 1;
switch (x) {
   case 1:
      System.out.println("Eins");
      break;
   case 2:
      System.out.println("Zwei");
      break;
}
```

# Sonderfall- bewusstes Durchlaufen

```
char note = 'A';


switch (note) {
   case 'A':
   case 'B':
      System.out.println("Bestanden");
      break;
   case 'F':
      System.out.println("Nicht bestanden");
      break;
   default:
      System.out.println("Ungültige Note");
}
```

case ‘A‘ und ‘B‘ werden gleich behandelt, da kein break dazwischen.

# Best Practices für switch

- Verwende immer **break**, außer du willst bewusst „durchlaufen“ lassen
- Füge **default immer hinzu,** um unerwartete Werte abzufangen.
- Halte den Code **übersichtlich** - jeder case sollte klar erkennbar sein.
- Verwende **switch** bei komplexen logischen Ausdrücken -> lieber if.

# Verständnisfragen zu default und break

- Was passiert, wenn in einem case kein break steht?
- Warum sollte man immer einen default Fall schreiben?
- Kann default auch am Anfang oder in der Mitte stehen?
- Was ist der Unterschied zwischen gewolltem und unbeabsichtigtem Fall-Through?
- Wie hilt break, Fehler zu vermeiden?